6 Messages

Overview

This section provides a technical description of messages which can be sent and received when using DSPower series software. Messages have been divided into two categories:

- library messages
- DSP Engine messages

Library messages are generated by DSPower series libraries (DLLs, VBXs, OCXs, etc.) and sent to windows or controls owned by calling applications or by DSPower-Block Diagram software. These messages are essentially relays, or "reformats", of information contained in messages received from DSP Engines. Currently, messages are not sent from calling applications to DSPower series libraries; instead, various functions available in the library APIs cause the libraries to generate and send messages to DSP Engines. Note that to receive messages of any type from DSPower series libraries, calling applications must first "register" one or more window or control handles using the DSRegisterEngineMsgWnd function in Engine Manager.

By contrast, messages involving DSP Engines are transmitted directly, in both directions, between DSP Engines and other applications. Because different DSP Engines have different means of communicating with the outside world, the Windows library approach

has been chosen to provide an intermediate layer and present a standardized programming interface. For example, Hypersignal-Macro series software sends information to a Windows virtual device driver (VxD), called Pipe.386, which arbitrates bi-directional shared memory command/data queues. Messages are formed as command and data information is extracted from these queues. As another example, MATLAB programs can send information using direct Windows API calls, via external library calls, or using a method based on the built-in MATLAB interactive control interface.

Thus, in DSPower series software, DSP Engine messages are normally processed automatically, without user-intervention. From the user's point of view--a C/C++ or MATLAB API in the case of DSPower series libraries, or a block diagram interface in the case of DSPower-Block Diagram software--these messages are not visible. However, DSP Engine messages have been fully documented to allow direct communication between user-defined applications and DSP Engines, if desired.

For general information on the rationale and benefits of the DSP Engine concept, see Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

Main Message Categories

Library Messages

DSP Engine Messages

6.1 Library Messages

The DSPower series library messages defined below can be received by calling applications which have implicitly or explicitly loaded Engine Manager (included with DSPower-HWLib and DSPower-SPLib software; see Signalogic DSP Software Users Guide section 7.1, Engine Manager Access). There are some important items to note about DSPower series librarymessages:

1. Calling applications must first "register" one or more window or control handles using the DSRegisterEngineMsgWnd function in Engine Manager, before receiving these messages.

2. All library messages are standard Windows "WM_" messages (declared using the WM_USER constant), and not Windows notification messages (signified, for example, by "EN_", "CBN_", etc.). Notification messages are not used because all messages contain one or more pieces of data.

3. All library messages begin with "WM_DSPENGINE_".

Library Messages

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_ENGINESTATE

WM_DSPENGINE_FLGRDY

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_LEVEL

WM_DSPENGINE_PROGSTATUS

WM_DSPENGINE_RUNTIMEERR

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_CODEGENERROR

hEngine = (HENGINE) wParam; // engine identifier

pErrorMsgInfo = (PERRORMSGINFO) lParam; // pointer to error information

The WM_DSPENGINE_CODEGENERROR engine status message is sent by Engine Manager after receiving an error message from a DSP

Engine which has encountered a format error or an in error in locating a native language source code file or executable file.

Parameter	Description
hEngine	handle of DSP Engine encountering code generation error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
short int	nErrorNum;	// error number or type
char	szErrorStatus[256];	// error/status string or error log filename

If nErrorNum is positive, then it represents a DSP Engine native language source or executable file processing error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently, negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_CODEGENERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_CODEGENRROR:
```

```
nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
if (nErrorNum >= 0) {

    _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

    _fstrcat(szErrorStatus, "; Error Number = ");
    char tmpstr[30];
    itoa(nErrorNum, tmpstr, 10);
    _fstrcat(szErrorStatus, tmpstr);

    MessageBox(NULL, szErrorStatus, "Code Generation Error Message",
    MB_OK);
}
else {

    _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
    hFile = OpenFile(szLogFile, &ofstruct);

    // process error log file...
}

break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_COMPILEERROR

```
hEngine = (HENGINE) wParam;          // engine identifier
pErrorMsgInfo = (PERRORMSGINFO)      // pointer to error information
lParam;
```

The WM_DSPENGINE_COMPILEERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine which has encountered an error compiling a native language source code file.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

```
// type        // element            // comment
short int      nErrorNum;           // error number or type
char           szErrorStatus[256];  // error/status string or error log filename
```

If nErrorNum is positive, then it represents a DSP Engine compilation error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
```

```
OFSTRUCT ofstruct;

case WM_DSPENGINE_COMPILEERROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Compile Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFNUM

hEngine = (HENGINE) wParam; // engine identifier
dwBufNum = (DWORD) lParam; // buffer number

The WM_DSPENGINE_BUFNUM engine update message is sent by Engine Manager after receiving a buffer number message from an engine executing a high-level data acquisition or generation function. The lParam parameter contains the current buffer number of the high-level generation function.

Parameter	Description
hEngine	handle of engine sending buffer number message
dwBufNum	current buffer number of high-level data acquisition or generation function

Returns

The calling application does not need to return a specific value.

Comments

The dwBufNum parameter indicates the current buffer number of the high-level data acquisition or generation function. Buffer numbers range from 0 to the maximum 32-bit word – 1, and increase monotonically at a constant rate dependent on the sampling frequency and buffer size being used in the high-level function. The current number of samples acquired (or generated) can be calculated by multiplying dwBufNum by the result obtained previously from a WM_DSPENGINE_BUFSIZE message.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_BUFNUM message being processed:

```
DWORD dwBufSize, dwBufNum;
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
```

```
        break;

case WM_DSPENGINE_BUFSIZE:

        dwBufSize = lParam;  // store the buffer length being used
        break;

case WM_DSPENGINE_BUFNUM:

        dwCurSamplePos = dwBufSize*lParam;  // calculate current sample count

        char tmpstr[30], tmpstr2[30];
        ltoa(dwCurSamplePos, tmpstr, 10);  // convert to text
        ftoa(dwCurSamplePos/Fs, tmpstr2, 10);

        UpdateStatusBar(hSampleCounter, tmpstr, "Current Sample Count");
        UpdateStatusBar(hTimeCounter, tmpstr2, "Current Time (seconds)");
        break;
```

See Also

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_BUFSIZE

hEngine = (HENGINE) wParam; // engine identifier

dwBufSize = (DWORD) lParam; // buffer size

The WM_DSPENGINE_BUFSIZE engine update message is sent by Engine Manager after receiving a buffer size message from a DSP Engine executing a high-level data acquisition, generation, or instrument function.

Parameter	Description
hEngine	handle of engine sending buffer size message
dwBufSize	buffer size being used during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The dwBufSize parameter indicates the buffer size being used during the high-level data acquisition, generation, or instrument function. Buffer sizes typically range from a few hundred samples up to 65535 samples.

Once the DSP Engine has calculated an appropriate buffer size, taking into account factors such as sampling rate, number of channels, and continuous signal generation requirements, the resulting value does not change for the duration of the high-level function. Note that in the case of continuous disk file acquisition or generation functions, it is typical for DSP Engines to calculate larger buffer sizes for high sampling rate values, in order to improve performance.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_BUFSIZE message being processed:

```
DWORD dwBufSize;
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    break;

case WM_DSPENGINE_BUFSIZE:
```

```
dwBufSize = lParam;  // store the buffer length being used

char tmpstr[30];
ftoa(dwBufSize/Fs, tmpstr, 10);

UpdateStatusBar(hTimeBase, tmpstr, "Framesize (in seconds)");
break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_BUFRDY

WM_DSPENGINE_BUFRDY

```
hEngine = (HENGINE) wParam;                    // engine identifier
pDSPDataRdyMsgInfo =                           // pointer to buffer information
(PDSPDATARDYMSGINFO) lParam;
```

The WM_DSPENGINE_BUFRDY engine callback message is sent by Engine Manager after receiving a buffer ready message from a DSP Engine which has acquired, generated, or processed a DSP/analog hardware data buffer.

Parameter	Description

hEngine	handle of DSP Engine encountering compiler error
pDSPDataRdyMsgInfo	pointer to PDSPDATARDYMSGINFO structure, which contains additional buffer information

Returns

The calling application does not need to return a specific value.

Comments

pDSPDataRdyMsgInfo points to a DSPDATARDYMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
short int	nNum;	// buffer or flag number
DWORD	dwFlagValue;	// flag value
DWORD	dwFlagAddr;	// flag address
HGLOBAL	hBoard;	// board handle

For more information, see section 5.1.3, DSPDATARDYMSGINFO Structure.

C/C++ Example

Below is an excerpt from the dscope.cpp source code example file (included with DSPower-HWLib software) showing a WM_DSPENGINE_BUFRDY message being received:

```
case WM_DSPENGINE_BUFRDY:  // buffer ready message sent by DSP engine

    pBuf = (int far*)GlobalLock(hBuf);  // lock memory to be used

    // transfer below illustrates two common type of DSP memory architectures

    if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

        if (nCurBuf == 0)  uStatus = DSGetMem(hBoard,
        DS_GM_VECTOR_DATA_X, dwTimBufferBaseAddr, DS_GM_SIZE16,
        pBuf, wBuflen);

        else  uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y,
        dwTimBufferBaseAddr, DS_GM_SIZE16, pBuf, wBuflen);
    }
    else { // linear data memory
```

```
        uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
        dwTimBufferBaseAddr+nCurBuf*Buflen, DS_GM_SIZE16, pBuf, wBuflen);
    }

    GlobalUnlock(hBuf);  // let memory float

    if (!uStatus) MessageBox(NULL, "DSGetMem problem", "Test Prog", MB_OK);

    // switch buffers and write new buffer number

    nCurBuf ^= 1;
    DSPutHVarMem(hBoard, DSP_HOSTBUFNUM, nCurBuf);

    // can we have another please, Ma'am..
    DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);
    break;
```

A significant amount of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the entire dscope.cpp file be examined before implementing code similar to the above example. For example, uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

In the above example, hBuf is assumed to have been created via a GlobalAlloc call, using a sizeof parameter equal to 2∗wBuflen. Memory pointers passed to DSGetMem and DSPutMem calls *must not point to memory segments less than the product of the dwCount and DS_GM_SIZExx parameters* used in the corresponding DSGetMem or DSPutMem calls. This product is in bytes.

See Also

WM_DSPENGINE_FLGRDY

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

Semaphore Messages from DSP Engines

WM_DSPENGINE_ENGINESTATE

WM_DSPENGINE_ENGINESTATE

```
hEngine = (HENGINE) wParam;          // engine identifier
wEngineState = LOWORD(lParam);       // current engine state
```

The WM_DSPENGINE_ENGINESTATE engine status message is sent by Engine Manager after receiving a status message from a DSP Engine indicating its current state, or when Engine Manager suspends or resumes an engine.

Parameter	Description
hEngine	handle of engine for which engine state message applies
wEngineState	current state of engine, as defined by one of the values below:

Value	Meaning
DS_ES_RUNNING	engine is in running (busy) state
DS_ES_IDLE	engine is in idle state
DS_ES_STOPPENDING	engine is running, but a stop (shutdown) command is pending
DS_ES_STOPPED	engine is stopped (shutdown)

Returns

The calling application does not need to return a specific value.

Comments

The wEngineState parameter reflects the current state of the DSP Engine. Note that receiving a WM_DSPENGINE_ENGINESTATE message does not necessarily mean there has been a state change. To detect changes, the calling application should update a local variable as part of its WM_DSPENGINE_ENGINESTATE message processing.

C/C++ Example

An example of processing a WM_DSPENGINE_ENGINESTATE message is shown below:

```
WORD static wPrevEngineState = DS_ES_IDLE;
WORD wEngineState;

case WM_DSPENGINE_ENGINESTATE:

    wEngineState = LOWORD(lParam);

    if (wEngineState == DS_ES_IDLE && wEngineState != wPrevEngineState) {

        UpdateStatusBar(hState, "Idle");  // engine state returned to idle
    }
    wPrevEngineState = wEngineState;  // update local copy of engine state
    break;
```

See Also

WM_DSPENGINE_PROGSTATUS

DSExecEngineFunction

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_FLGRDY

WM_DSPENGINE_FLGRDY

```
hEngine = (HENGINE) wParam;                    // engine identifier
pDSPDataRdyInfo =                              // pointer to flag information
(PDSPDATARDYMSGINFO) lParam;
```

The WM_DSPENGINE_FLGRDY engine callback message is sent by Engine Manager after receiving a buffer ready message from a DSP Engine which has acquired, generated, or processed a DSP/analog hardware data buffer.

Parameter	Description

hEngine	handle of DSP Engine encountering compiler error
pDSPDataRdyInfo	pointer to PDSPDATARDYMSGINFO structure, which contains additional flag information

Returns

The calling application does not need to return a specific value.

Comments

pDSPDataRdyMsgInfo points to a DSPDATARDYMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
short int	nNum;	// buffer or flag number
DWORD	dwFlagValue;	// flag value
DWORD	dwFlagAddr;	// flag address
HGLOBAL	hBoard;	// board handle

For more information, see section 5.1.3, DSPDATARDYMSGINFO Structure.

C/C++ Example

Below is an example showing a WM_DSPENGINE_FLGRDY message being received:

```
case WM_DSPENGINE_FLGRDY:  // semaphore message sent by DSP engine

    DWORD dwCount = 4096;  // process 4096 floating-point values each frame

    HGLOBAL hBuf = GlobalAlloc(GMEM_INIT | GMEM_MOVEABLE,
    dwCount*DS_GM_SIZE32);

    (float far*) pBuf = (float far*)GlobalLock(hBuf);  // lock memory before use

    // transfer below illustrates two common type of DSP memory architectures

    if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

        uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X,
        dwResultDataAddr, DS_GM_SIZE32, pBuf, dwCount);
    }
    else { // linear data memory
```

```
        uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT,
        dwResultDataAddr, DS_GM_SIZE32, pBuf, dwCount);
    }

    GlobalUnlock(hBuf);  // let memory go

    if (!uStatus) {

        // error occurred in DSGetMem call
        MessageBox(NULL, "DSGetMem problem", "Example", MB_OK);
    }
    else {

        // wait for semaphore again...
        DSWaitForFlag(hBoard, 0, NULL, DS_WFB_POLLED);
    }
    break;
```

A great deal of initialization is necessary before buffer ready and semaphore messages can be processed. The example above leaves out much of this; it is strongly recommended that the dscope.cpp example source code file (included with DSPower-HWLib software) be examined as at least one good illustration before implementing code similar to the above example. For example, uMemArch is assumed to have been initialized beforehand using the DSGetMemArch call.

In the above example, note that hBuf is created via a GlobalAlloc call, using a sizeof parameter equal to the total element count∗memory width per element. Memory pointers passed to DSGetMem and DSPutMem calls *must not point to memory segments less than the product of the dwCount and DS_GM_SIZExx parameters* used in the corresponding DSGetMem or DSPutMem calls. The resulting product is in bytes.

See Also

WM_DSPENGINE_BUFRDY

DSWaitForBuffer

DSWaitForFlag

DSRegisterEngineMsgWnd

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Buffer Ready Messages from DSP Engines

WM_DSPENGINE_FUNCTIONERROR

WM_DSPENGINE_FUNCTIONERROR

hEngine = (HENGINE) wParam; // engine identifier

pErrorMsgInfo = (PERRORMSGINFO) lParam; // pointer to error information

The WM_DSPENGINE_FUNCTIONERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine for which a function, display, or instrument has encountered an input parameters or other setup error. This error message is sent before the function starts.

Parameter	Description
hEngine	handle of DSP Engine encountering function error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

The pErrorMsgInfo parameter points to an ERRORMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
short int	nErrorNum;	// error number or type
char	szErrorStatus[256];	// error/status string or error log filename

If nErrorNum is positive, then it represents a DSP Engine function initialization error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently,

negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_COMPILEERROR message being received:

```
char szErrorStatus[256];
char szLogFile[256];
HFILE hFile;
OFSTRUCT ofstruct;

case WM_DSPENGINE_FUNCTIONERROR:

    nErrorNum = (PERRORMSGINFO)lParam->nErrorNum;
    if (nErrorNum >= 0) {

        _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);

        _fstrcat(szErrorStatus, "; Error Number = ");
        char tmpstr[30];
        itoa(nErrorNum, tmpstr, 10);
        _fstrcat(szErrorStatus, tmpstr);

        MessageBox(NULL, szErrorStatus, "Function Error Message", MB_OK);
    }
    else {

        _fstrcpy(szLogFile, PERRORMSGINFO)lParam->szErrorStatus);
        hFile = OpenFile(szLogFile, &ofstruct);

        // process error log file...
    }

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_RUNTIMEERROR

DSRegisterEngineMsgWnd

Related Users Guide Topics

Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_LEVEL

WM_DSPENGINE_LEVEL

```
hEngine = (HENGINE) wParam;      // engine identifier
dwLevel = (DWORD) lParam;        // maximum amplitude level
```

The WM_DSPENGINE_LEVEL engine update message is sent by Engine Manager when a DSP Engine amplitude level message is received during a high-level data acquisition, generation, or instrument function. The lParam parameter contains the maximum absolute amplitude found in each buffer of acquired or generated data.

Parameter	Description
hEngine	handle of engine sending level message
dwLevel	maximum amplitude level found in each buffer during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The dwLevel parameter indicates the maximum absolute value found in each buffer of data processed during the high-level data acquisition, generation, or instrument function. Values range from 0 to the maximum full-scale value of the A/D or D/A converter(s) being used in the high-level function.

Note that values are always positive.

C/C++ Example

Below is an example of processing a WM_DSPENGINE_LEVEL message:

```
case WM_DSPENGINE_LEVEL:

    UpdateMeter(hMeter, lParam);  // lParam has maximum amplitude
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_SAMPFREQ

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_PROGSTATUS

WM_DSPENGINE_PROGSTATUS

```
hEngine = (HENGINE) wParam;          // engine identifier
wProgramStatus = LOWORD(lParam);     // program status
wFunctionID = HIWORD(lParam);        // function ID
```

The WM_DSPENGINE_PROGSTATUS program status message is sent by Engine Manager after receiving a program status message from a DSP Engine containing status information regarding an executing program.

Parameter	Description
hEngine	handle of engine for which program status information applies
wProgramStatus	program status, as defined by one of the values below:

Value	Meaning
DS_PS_FUNCDONE	engine function is complete
DS_PS_PROGDONE	program is complete

wFunctionID	identifier of function, if wProgramStatus above indicates a function has completed

Returns

The calling application does not need to return a specific value.

Comments

Depending on its flow of execution, a DSP Engine program may cause several DSP Engine functions to execute. Processing WM_DSPENGINE_PROGSTATUS functions provides a method to dynamically monitor the status of DSP Engine programs, regardless of the execution variations which are data-dependent or user-dependent.

Note that function ID values must be encoded into the source code generated and submitted to the DSP Engine to be executed as a native language program. This is done using the SetFunctionID DSP Engine function, which sets the current function ID for use in DSP Engine communication with the outside world.

Note that WM_DSPENGINE_PROGSTATUS program status callback messages are sent before any library functions, such as DSSendEngineCommand or DSExecEngineFunction,. which might be waiting for "program done" notification, are released. This situation can occur if these functions were executed by the calling application in asynchronous mode (uMode parameter == DS_EEF_ASYNC).

C/C++ Example

An example of processing a WM_DSPENGINE_PROGSTATUS message is shown below:

```
case WM_DSPENGINE_PROGSTATUS:

    if (LOWORD(lParam) == DS_PS_PROGDONE) {

        UpdateStatusBar(hProgStatus, "Engine Program Complete");
    }
    break;
```

See Also

DSRegisterEngineMsgWnd

DSExecEngineFunction

WM_DSPENGINE_ENGINESTATE

Related Users Guide Topics

Program Status Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_RUNTIMEERROR

WM_DSPENGINE_RUNTIMEERROR

hEngine = (HENGINE) wParam; // engine identifier

pErrorMsgInfo = (PERRORMSGINFO) lParam; // pointer to error information

The WM_DSPENGINE_RUNTIMEERROR engine status message is sent by Engine Manager after receiving an error message from a DSP Engine which has encountered an error while executing a program.

Parameter	Description
hEngine	handle of DSP Engine encountering compiler error
pErrorMsgInfo	pointer to PERRORMSGINFO structure, which contains additional error information

Returns

The calling application does not need to return a specific value.

Comments

pErrorMsgInfo points to an ERRORMSGINFO structure, which can be summarized as follows:

// type	// element	// comment
short int	nErrorNum;	// error number or type
char	szErrorStatus[256];	// error/status string or error log filename

If nErrorNum is positive, then it represents a DSP Engine run-time error number (as defined in DSP Engine documentation) and szStatusString is an error condition string. If the nErrorNum parameter is negative, then szStatusString is a pathname for an error log file. Currently, for DSPower series software-compatible engines, error log files are only produced during compilation (i.e. currently, negative nErrorNum values only apply to WM_DSPENGINE_COMPILEERROR messages). For more information, see section 5.1.4, ERRORMSGINFO Structure.

C/C++ Example

Below is a partial source code example showing a WM_DSPENGINE_RUNTIMEERROR message being received:

```
char szErrorStatus[256], char tmpstr[30];

case WM_DSPENGINE_RUNTIMEERROR:

    _fstrcpy(szErrorStatus, (PERRORMSGINFO)lParam->szErrorStatus);
    _fstrcat(szErrorStatus, "; Error Number = ");
    itoa((PERRORMSGINFO)lParam->nErrorNum, tmpstr, 10);
    _fstrcat(szErrorStatus, tmpstr);

    MessageBox(NULL, szErrorStatus, "Run-Time Error Message", MB_OK);

    break;
```

Note if multiple engines are open, the wParam parameter can be used to identify which engine has sent the error message.

See Also

WM_DSPENGINE_CODEGENERROR

WM_DSPENGINE_COMPILEERROR

WM_DSPENGINE_FUNCTIONERROR

DSRegisterEngineMsgWnd

DSExecEngineFunction

Related Users Guide Topics

section 7.4 Registering Callback Messages from DSP Engines

Status Messages from DSP Engines

section 7.3 Executing Functions in DSP Engines

WM_DSPENGINE_SAMPFREQ

WM_DSPENGINE_SAMPFREQ

hEngine = (HENGINE) wParam; // engine identifier

SampFreq = (float) lParam; // sampling frequency

The WM_DSPENGINE_SAMPFREQ engine update message is sent by Engine Manager when a DSP Engine "actual sampling frequency" message is received during a high-level data acquisition, generation, or instrument function. The lParam parameter contains the actual sampling frequency being used during the function.

Parameter	Description
hEngine	handle of DSP Engine sending sampling frequency message
SampFreq	sampling frequency being used during high-level data acquisition, generation or instrument function

Returns

The calling application does not need to return a specific value.

Comments

The SampFreq parameter indicates the sampling frequency, in Hz, being used during the high-level data acquisition, generation, or instrument function. Sampling frequency values typically range from a few Hz up to 10 MHz or higher.

Although all high-level DSP Engine functions include a desired sampling rate parameter, the actual rate used may differ. This occurs because many types of DSP/analog hardware offer only a limited number of discrete sampling rate values. DSP Engine high-level functions then choose the physical value closest to the desired value, and report the choice using the WM_DSPENGINE_SAMPFREQ message.

C/C++ Example

Below is an example which shows a WM_DSPENGINE_SAMPFREQ message being processed:

```
float Fs;

case WM_DSPENGINE_SAMPFREQ:

    _fmemcpy(&Fs, &lParam, sizeof(float));  // store actual Fs being used
    char tmpstr[30];
    gcvt(Fs, 10, tmpstr);

    UpdateStatusBar(hSampFreq, tmpstr, "Sampling Frequency (in Hz)");
    break;
```

See Also

WM_DSPENGINE_BUFNUM

WM_DSPENGINE_BUFSIZE

WM_DSPENGINE_LEVEL

DSAcquireWvfrmFile

DSGenerateWvfrmFile

DSExecEngineFunction

Related Users Guide Topics

Update Messages from DSP Engines in section 7.4, Registering Callback Messages from DSP Engines

section 4.5 Acquiring Data Using DSPower-HWLib Software

section 7.3 Executing Functions in DSP Engines

6.2 DSP Engine Messages

This section documents messages sent by DSP Engines. Normally, in DSPower series software, these messages are processed automatically, without user-intervention. For example, in response to a DSEngineOpen call, the DSPower-HWLib library receives and sends engine messages to Hypersignal-Macro series software in order to initialize the software in engine mode. From the user's point of view, in this case a C/C++ or MATLAB API, these messages are not visible--only the library call, in this case DSEngineOpen, is visible. Another example is the DSPower-Block Diagram package, which either processes DSP Engine messages directly or uses an intermediate layer such as the DSPower-HWLib library. Again, from the user's point of view, engine messages are not visible--instead, an easy-to-use, graphical block diagram user-interface is visible.

For more information on the rationale and benefits of the DSP Engine concept, see Signalogic DSP Software System Guide section 1.1, Relationship of Signalogic DSP Software Components.

The actual numeric constants defined for DSP Engine messages are contained in the include files enmgr.h and engdef.h, which are supplied with DSPower-HWLib and DSPower-SPLib software, and which may be more current than this documentation.

DSP Engine Messages

DSPENG_ABORT

DSPENG_BUFRDY

DSPENG_CALCSAMPFREQ

DSPENG_CALCSAMPFREQRESULT

DSPENG_CANCELWAIT

DSPENG_CLIPRECT

DSPENG_CODEGENERROR

DSPENG_COMPILEERROR

DSPENG_DISABLEBOARD

DSPENG_DISPDIRFILESTEXT

DSPENG_DISPSTATUSTEXT

DSPENG_DISPWVFRMSUMTEXT

DSPENG_ERASERECT

DSPENG_ESC

DSPENG_EXIT

DSPENG_FLGRDY

DSPENG_FRAMERECT

DSPENG_FUNCTIONACTIVE

DSPENG_FUNCTIONDONE

DSPENG_FUNCTIONERROR

DSPENG_FUNCTIONPAUSE

DSPENG_GETBOARDCLASS

DSPENG_GETBOARDCLASSRESULT

DSPENG_GETHVARMEM

DSPENG_GETHVARMEMRESULT

DSPENG_GETMEM

DSPENG_GETMEMRESULT

DSPENG_GETMEMSIZE

DSPENG_GETMEMSIZERESULT

DSPENG_GETPARMS

DSPENG_HOLDBOARD

DSPENG_INITBOARD

DSPENG_INITBOARDSTATUS

DSPENG_LINEREL

DSPENG_LINETO

DSPENG_LOADFILE

DSPENG_LOADFILESTATUS

DSPENG_MOVEREL

DSPENG_MOVETO

DSPENG_PAINTRECT

DSPENG_PUTHVARMEM

DSPENG_PUTMEM

DSPENG_PUTMEMRESULT

DSPENG_RDBOARDREG

DSPENG_RDBOARDREGRESULT

DSPENG_REDRAW

DSPENG_REFRESH

DSPENG_REPORTBUFNUM

DSPENG_REPORTBUFSIZE

DSPENG_REPORTDISPID

DSPENG_REPORTFUNCID

DSPENG_REPORTLEVEL

DSPENG_REPORTSAMPFREQ

DSPENG_RESETBOARD

DSPENG_RUNBOARD

DSPENG_RUNEXTPROGASYNC

DSPENG_RUNEXTPROGSYNC

DSPENG_RUNTIMEERROR

DSPENG_SETAMP

DSPENG_SETBACKCOLOR

DSPENG_SETBITMAP

DSPENG_SETBOARDBASEADDR

DSPENG_SETBUFNUMID

DSPENG_SETCOLOR

DSPENG_SETCURTRACE

DSPENG_SETDISPID

DSPENG_SETDISPLAY

DSPENG_SETFUNCID

DSPENG_SETGCUR

DSPENG_SETLEVELID

DSPENG_SETOFS

DSPENG_SETPENCOLOR

DSPENG_SETPENMODE

DSPENG_SETRES

DSPENG_SETPIXEL

DSPENG_SETSTATISTICSID

DSPENG_SETTRACE

DSPENG_SETUNITS

DSPENG_SETUP

DSPENG_SETVOLTAGERANGE

DSPENG_SHUTDOWN

DSPENG_WAITFORBUF

DSPENG_WAITFORFLG

DSPENG_WAVEFORMEDITED

DSPENG_WRBOARDREG

WM_DSPENGINE_CMDDATA

DSPENG_ABORT

// command instructing engine to abort and unload from system, without saving currently open files or configuration

DSPENG_BUFRDY

// engine is indicating that buffer previously assigned by DSPENG_WAITFORBUF command is ready

DSPENG_CALCSAMPFREQ

// command instructing engine to calculate actual sampling frequency for specified DSP/analog hardware

DSPENG_CALCSAMPFREQRESULT

// engine is sending result of DSPENG_CALCSAMPFREQ command

DSPENG_CANCELWAIT

// command instructing engine to cancel pending wait requests

DSPENG_CLIPRECT

// engine is sending command to set clipping rectangle

DSPENG_CODEGENERROR

// engine encountered code generation error (WORD error number)

DSPENG_COMPILEERROR

// engine encountered compile error (WORD error number)

DSPENG_DISABLEBOARD

// command instructing engine to disable specified DSP/analog hardware

DSPENG_DISPDIRFILESTEXT

// engine is sending file directory status for display

DSPENG_DISPSTATUSTEXT

// engine is sending status text for display

DSPENG_DISPWVFRMSUMTEXT

// engine is sending waveform summary text for display

DSPENG_ERASERECT

// engine is sending command to erase rectangle

DSPENG_ESC

// command instructing engine to escape back up one level

DSPENG_EXIT

// command instructing engine to save open data and configuration files and unload from system

DSPENG_FLGRDY

// engine is indicating that flag previously assigned by DSPENG_WAITFORFLG command is ready

DSPENG_FRAMERECT

// engine is sending command to draw rectangle frame

DSPENG_FUNCTIONACTIVE

// engine function or program active

DSPENG_FUNCTIONDONE

// engine function or program complete

DSPENG_FUNCTIONERROR

// engine function or program terminated with error

DSPENG_FUNCTIONPAUSE

// engine function or program has paused

DSPENG_GETBOARDCLASS

// command instructing engine to return driver classification for specified DSP/analog hardware

DSPENG_GETBOARDCLASSRESULT

// engine is sending result of DSPENG_GETBOARDCLASS command

DSPENG_GETHVARMEM

// command instructing engine to read Hypersignal DSP Source Code Interface variable

DSPENG_GETHVARMEMRESULT

// engine is sending result of DSPENG_GETHVARMEM command

DSPENG_GETMEM

// command instructing engine to read data from DSP memory; for specified DSP/analog hardware

DSPENG_GETMEMRESULT

// engine is reporting completion status of previous DSPENG_GETMEM command

DSPENG_GETMEMSIZE

// command instructing engine to calculate maximum data memory size for specified DSP/analog hardware

DSPENG_GETMEMSIZERESULT

// engine is sending result of DSPENG_GETMEMSIZE size command

DSPENG_GETPARMS

// engine can accept parameters

DSPENG_HOLDBOARD

// command instructing engine to place specified DSP/analog hardware in hold mode

DSPENG_INITBOARD

// command instructing engine to initialize specified DSP/analog hardware

DSPENG_INITBOARDSTATUS

// engine is reporting result of DSPENG_INITBOARD command (DSP/analog hardware initialization status)

DSPENG_LINEREL

// engine is sending command to draw line relative

DSPENG_LINETO

// engine is sending command to draw line absolute

DSPENG_LOADFILE

// command instructing engine to download DSP Program File for specified DSP/analog hardware

DSPENG_LOADFILESTATUS

// engine is reporting completion status of DSPENG_LOADFILE command

DSPENG_MOVEREL

// engine is sending command to move graphics position relative

DSPENG_MOVETO

// engine is sending command to move graphics position absolute

DSPENG_PAINTRECT

// engine is sending command to paint rectangle

DSPENG_PUTHVARMEM

// command instructing engine to write Hypersignal DSP Source Code Interface variable

DSPENG_PUTMEM

// command instructing engine to write data to DSP memory for specified DSP/analog hardware

DSPENG_PUTMEMRESULT

// engine is reporting completion status of previous DSPENG_PUTMEM command

DSPENG_RDBOARDREG

// command instructing engine to read board register for specified DSP/analog hardware

DSPENG_RDBOARDREGRESULT

// engine is sending result of DSPENG_RDBOARDREG command

DSPENG_REDRAW

// command instructing engine to return to initial frame parameters

DSPENG_REFRESH

// command instructing engine to refresh display

DSPENG_REPORTBUFNUM

// engine is reporting current DSP buffer count for high-level function

DSPENG_REPORTBUFSIZE

// engine is reporting DSP buffer size being used in high-level function

DSPENG_REPORTDISPID

// engine is reporting new active display ID

DSPENG_REPORTFUNCID

// engine is reporting new active function ID

DSPENG_REPORTLEVEL

// engine is reporting current maximum absolute amplitude level for data buffer in high-level function

DSPENG_REPORTSAMPFREQ

// engine is reporting DSP/analog hardware sampling frequency being used in high-level function

DSPENG_RESETBOARD

// command instructing engine to reset specified DSP/analog hardware

DSPENG_RUNBOARD

// command instructing engine to run specified DSP/analog hardware

DSPENG_RUNEXTPROGASYNC

// run external program asynchronously with engine

DSPENG_RUNEXTPROGSYNC

// run an external program synchronously with engine

DSPENG_RUNTIMEERROR

// engine encountered run-time error (WORD error number)

DSPENG_SETAMP

// engine is sending command to set amplitude "hotspot" area

DSPENG_SETBACKCOLOR

// engine is sending command to set background color

DSPENG_SETBITMAP

// engine is sending command to set current bitmap

DSPENG_SETBOARDBASEADDR

// command instructing engine to set I/O and memory base addresses
for specified DSP/analog hardware

DSPENG_SETBUFNUMID

// command instructing engine to set buffer number message ID

DSPENG_SETCOLOR

// command instructing engine to set current drawing color

DSPENG_SETCURTRACE

// engine requesting to set current display trace in multitrace display

DSPENG_SETDISPID

// command instructing engine to set current display ID

DSPENG_SETDISPLAY

// engine is sending command to set current display page

DSPENG_SETFUNCID

// command instructing engine to set current function ID

DSPENG_SETGCUR

// command instructing engine to set current graphics cursor

DSPENG_SETLEVELID

// command instructing engine to set amplitude level message ID

DSPENG_SETOFS

// command instructing engine to set current graphics offset

DSPENG_SETPENCOLOR

// engine is sending command to set pen color

DSPENG_SETPENMODE

// engine is sending command to set pen mode

DSPENG_SETRES

// command instructing engine to set current graphics resolution

DSPENG_SETPIXEL

// engine is sending command to set pixel

DSPENG_SETSTATISTICSID

// command instructing engine to set statistics message ID

DSPENG_SETTRACE

// engine is sending command to set current trace "hotspot" area

DSPENG_SETUNITS

// engine is sending command to set units "hotspot" area

DSPENG_SETUP

// engine sends this message after initializing and entering command mode

DSPENG_SETVOLTAGERANGE

// command instructing engine to set voltage ranges for specified DSP/analog hardware

DSPENG_SHUTDOWN

// engine sends this command as last action before shutdown

DSPENG_WAITFORBUF

// command instructing engine to assign wait-for-buffer request

DSPENG_WAITFORFLG

// command instructing engine to assign wait-for-flag request

DSPENG_WAVEFORMEDITED

// engine reports waveform has been edited

DSPENG_WRBOARDREG

// command instructing engine to write board register for specified DSP/analog hardware

WM_DSPENGINE_CMDDATA

WM_DSPENGINE_CMDDATA

wMsgType = wParam; // message type

pEngineSharedInfo = (PENGINESHAREDINFO) lParam; // pointer to shared memory

The WM_DSPENGINE_CMDDATA message is sent by DSP Engines to external applications. Depending on the engine, it can be sent using a VxD or using direct Windows API SendMessage, PostMessage, or PostAppMessage calls.

Parameter	Description
wMsgType	message type, as defined below:

Value	Meaning
DS_CD_VXDMSG	message sent via VxD
DS_CD_APIMSG	message sent from standard API call

pEngineSharedInfo	pointer to shared memory, which can be accessed receiving application using ENGINESHAREDINFO structure definition in engdef.h

Returns

The calling application does not need to return a specific value.

Comments

The WM_DSPENGINE_CMDDATA message contains a pointer to a shared memory area. Normally, this shared area contains bi-directional queues in which the external application and DSP Engine can exchange command and data information at maximum transfer rates. The bi-directional queues allow exchange of "command packets", which contain variable length data elements depending on the nature of the command. It is these commands which form many of the possible DSP Engine messages.

C/C++ Example

An example of processing a WM_DSPENGINE_CMDDATA message is shown below:

```
case WM_DSPENGINE_CMDDATA:

    pSharedEngineInfo = (PENGINESHAREDINFO)lParam;

    while (!IsQueueEmpty()) {

        Cmd = GetQueueByte();

        switch (Cmd) {

            // process commands
        }
    }
    break;
```

See Also

ENGINESHAREDINFO

DSExecEngineFunction

Related Users Guide Topics

section 7.3 Executing Functions in DSP Engines

Notes: